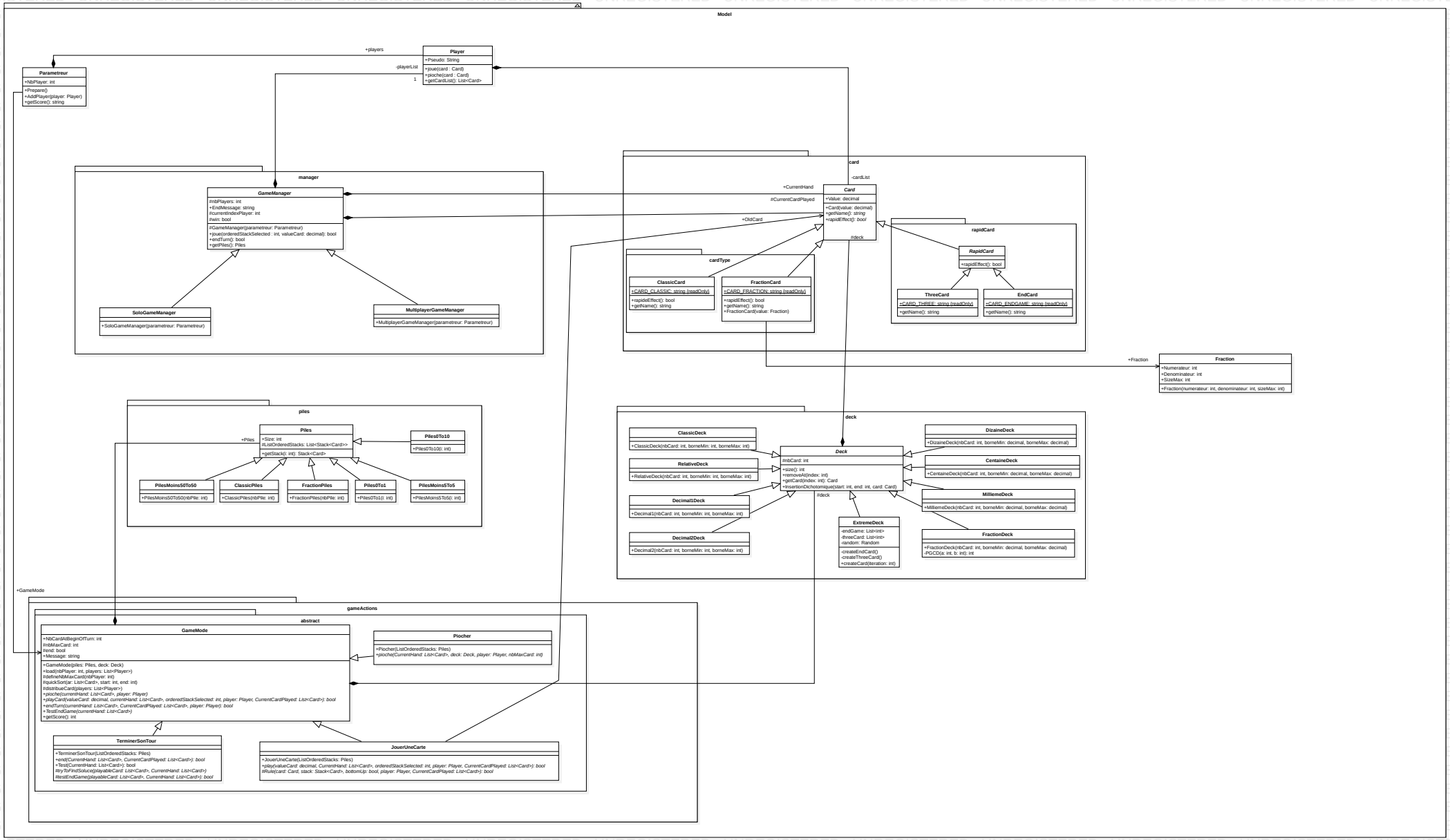
Model
Parametreur
+NbPlayer: int
+Prepare()
+AddPlayer(player: Player)
+getScore(): string
+players
Player
+Pseudo: String
+joue(card : Card)
+ploche(card : Card)
+getCardList(): List<Card>
-playerList
1
manager
GameManager
#nbPlayers: int
+EndMessage: string
#currentIndexPlayer: int
#win: bool
#GameManager(parametreur: Parametreur)
+joue(orderedStackSelected : int, valueCard: decimal): bool
+endTurn(): bool
+getPiles(): Piles
SoloGameManager
+SoloGameManager(parametreur: Parametreur)
MultiplayerGameManager
+MultiplayerGameManager(parametreur: Parametreur)
card
-cardList
+CurrentHand
#CurrentCardPlayed
Card
+Value: decimal
+Card(value: decimal)
+getName(): string
+rapidEffect(): bool
+OldCard
#deck
cardType
ClassicCard
+CARD_CLASSIC: string {readOnly}
+rapideEffect(): bool
+getName(): string
FractionCard
+CARD_FRACTION: string {readOnly}
+rapidEffect(): bool
+getName(): string
+FractionCard(value: Fraction)
rapidCard
RapidCard
+rapidEffect(): bool
ThreeCard
+CARD_THREE: string {readOnly}
+getName(): string
EndCard
+CARD_ENDGAME: string {readOnly}
+getName(): string
+Fraction
Fraction
+Numerateur: int
+Denominateur: int
+SizeMax: int
+Fraction(numerateur: int, denominateur: int, sizeMax: int)
piles
Piles
+Size: int
#ListOrderedStacks: List<Stack<Card>>
+getStack(i: int): Stack<Card>
+Piles
Piles0To10
+Piles0To10(i: int)
PilesMoins50To50
+PilesMoins50To50(nbPile: int)
ClassicPiles
+ClassicPiles(nbPile: int)
FractionPiles
+FractionPiles(nbPile: int)
Piles0To1
+Piles0To1(i: int)
PilesMoins5To5
+PilesMoins5To5(i: int)
deck
ClassicDeck
+ClassicDeck(nbCard: int, borneMin: int, borneMax: int)
RelativeDeck
+RelativeDeck(nbCard: int, borneMin: int, borneMax: int)
Decimal1Deck
+Decimal1(nbCard: int, borneMin: int, borneMax: int)
Decimal2Deck
+Decimal2(nbCard: int, borneMin: int, borneMax: int)
Deck
#nbCard: int
+size(): int
+removeAt(index: int)
+getCard(index: int): Card
+InsertionDichotomique(start: int, end: int, card: Card)
#deck
DizaineDeck
+DizaineDeck(nbCard: int, borneMin: decimal, borneMax: decimal)
CentaineDeck
+CentaineDeck(nbCard: int, borneMin: decimal, borneMax: decimal)
MilliemeDeck
+MilliemeDeck(nbCard: int, borneMin: decimal, borneMax: decimal)
FractionDeck
+FractionDeck(nbCard: int, borneMin: decimal, borneMax: decimal)
-PGCDa: int, b: int): int
ExtremeDeck
-endGame: List<int>
-threeCard: List<int>
-random: Random
-createEndCard()
-createThreeCard()
+createCard(iteration: int)
+GameMode
gameActions
abstract
GameMode
+NbCardAtBeginOfTurn: int
#nbMaxCard: int
#end: bool
+Message: string
+GameMode(piles: Piles, deck: Deck)
+load(nbPlayer: int, players: List<Player>)
#defineNbMaxCard(nbPlayer: int)
#quickSort(ar: List<Card>, start: int, end: int)
#distribueCard(players: List<Player>)
+ploche(currentHand: List<Card>, player: Player)
+playCard(valueCard: decimal, currentHand: List<Card>, orderedStackSelected: int, player: Player, CurrentCardPlayed: List<Card>): bool
+endTurn(currentHand: List<Card>, CurrentCardPlayed: List<Card>, player: Player): bool
+TestEndGame(currentHand: List<Card>)
+getScore(): int
Plocher
+Plocher(ListOrderedStacks: Piles)
+ploche(CurrentHand: List<Card>, deck: Deck, player: Player, nbMaxCard: int)
TerminerSonTour
+TerminerSonTour(ListOrderedStacks: Piles)
+end(CurrentHand: List<Card>, CurrentCardPlayed: List<Card>): bool
+Test(CurrentHand: List<Card>): bool
#tryToFindSoluce(playableCard: List<Card>, CurrentHand: List<Card>)
#testEndGame(playableCard: List<Card>, CurrentHand: List<Card>): bool
JouerUneCarte
+JouerUneCarte(ListOrderedStacks: Piles)
+play(valueCard: decimal, CurrentHand: List<Card>, orderedStackSelected: int, player: Player, CurrentCardPlayed: List<Card>): bool
#Rule(card: Card, stack: Stack<Card>, bottomUp: bool, player: Player, CurrentCardPlayed: List<Card>): bool
+GameMode